AP-712

APPLICATION NOTE

DRAM Controller for i960® JA/JF/JD Microprocessors

Paul Durazo
SPG EPD 80960 Applications Engineer

Intel Corporation
Embedded Processor Division
Mail Stop CH5-233
5000 W. Chandler Blvd.
Chandler, Arizona 85226

February 9, 1995

Order Number: 272674-001

Additional copies of this document or other Intel literature may be obtained from:

Intel Corporation
Literature Sales
P.O. Box 7641
Mt. Prospect, IL 60056-7641

or call 1-800-879-4683

DRAM CONTROLLER FOR I960® JA/JF/JD MICROPROCESSORS

FIGURES

TABLES

1.0 INTRODUCTION

This application note describes a DRAM controller for use with Intel's i960® JA/JF/JD microprocessors. Other application notes are available which describe DRAM controllers for the i960 CA and CF processors; see Section 6.0, RELATED INFORMATION for ordering information.

This DRAM controller's design features include:

- Non-interleaved design
- Can use standard 70 ns DRAM SIMM
- 3-1-1-1 wait state burst reads at 33 MHz
- 2-1-1-1 wait state burst writes at 33 MHz
- No delay lines

This application note contains some general DRAM controller theory as well as this design's state machine definitions and timing diagrams. It also contains the PLD equations which were used to build and test the prototype design. Timing analysis was verified with Timing Designer*. PLD equations were created in ABEL* as a device-independent design. Schematics were developed with OrCAD*. The timing analysis, schematics and PLD files are available through Intel's America's Application Support BBS, at (916) 356-3600.

1.1 Design Goals

A primary goal was to implement a single or dual bank 32-bit DRAM controller with the minimum number of components, using a standard 72-pin fast page mode DRAM SIMM. Such a design may be useful in embedded systems where space is at a premium. Accordingly, this design avoids such techniques as write posting, parity support and bank interleaving. A non-interleaved design significantly reduces system cost and complexity by using fewer components and logic. The memory in this design can be divided into two banks for addressing flexibility.

1.2 i960 Jx Processor Bus/Core Frequencies

This DRAM controller is designed for use with a 5 volt i960 Jx processor with bus frequencies up to 40 MHz. The design was verified with a 33 MHz version (production 40 MHz devices were not available at the time this document was published).

Available 80960 JA/JF speeds are 16 MHz, 25 MHz and 33 MHz. Intel's clock-doubled version, the 80960JD, operates at 16.7 MHz, 20 MHz and 25 MHz bus speeds, with the core operating at 33.3 MHz, 40 MHz and 50 MHz, respectively. Table 1 details the various DRAM speeds, bus speeds and wait state combinations that were tested with this design.

1.3 Page Mode DRAM SIMM

Page mode DRAM allows faster memory access by keeping the same row address while selecting random column addresses within that row. A new column address is selected by deasserting $\overline{CAS}$ while keeping $\overline{RAS}$ active and then asserting $\overline{CAS}$ with the new column address valid to the DRAM. Page mode operation works very well with burst buses, such as the i960 CA/CF processor bus, in which a single address cycle can be followed by up to four data cycles.

All $\overline{WE}$ pins on the SIMM are tied to a common $\overline{WE}$ line; this feature requires the use of early write cycles. In an early write cycle, write data is referenced to the falling edge of $\overline{CAS}$, not the falling edge of $\overline{WE}$.

The DRAM SIMM has four $\overline{CAS}$ lines, one for each eight (nine) bits in a 32-bit (36-bit) SIMM module. The four $\overline{CAS}$ lines control the writing to individual bytes within the SIMM. Each $\overline{CAS}$ signal is asserted with its respective $\overline{BE}$.

1.4 Burst Capabilities for 32-Bit Bus

The i960 Jx processor can access up to four data words per request. A request starts with the processor asserting $\overline{ADS}$ in the address cycle, and ends by the processor asserting $\overline{BLAST}$ in the last data cycle. Figure 1 shows $\overline{ADS}$ and $\overline{BLAST}$ timings for a quad-word write request.

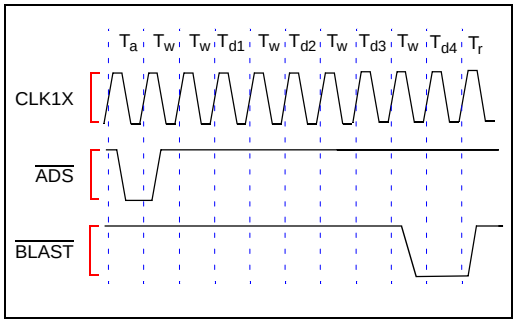

Figure 1. Quad-Word Write Request with 2,1,1,1 Wait State Profile

The processor's burst capabilities on a 32-bit bus include:

- Quad-word and triple-word requests start on quad-word boundaries (A3 = 0, A2 = 0).
- Double-word requests start on double-word boundaries (A3 = X, A2 = 0).
- Single-word requests can start on any word boundary (A3 = X, A2 = X).
- Any request starting on an odd word boundary never bursts (A3 = X, A2 = 1).

2.0 BASIC DRAM CONTROLLER

The main state machine of the DRAM controller in this design is clocked using CLKIN (CLK1X clock). The DRAM region is set at 0xAxxxxxxx for address decoding and uses the entire 256 Mbyte region. The region must also be configured as 32 bit in the processor's PMCON10_11 register. The controller is implemented as a four-bit state machine and is responsible for sequencing accesses as well as refreshes to the DRAM banks. Figure 2 shows the basic DRAM controller state machine.

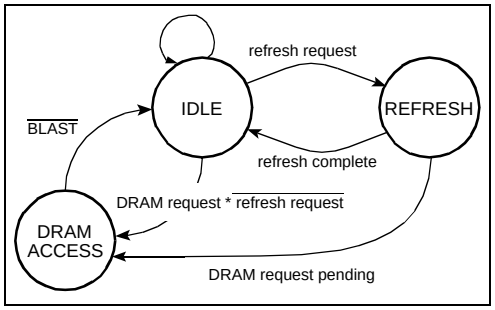

Figure 2. Basic DRAM Controller State Machine

The first state is IDLE. The state machine transitions from the IDLE state based on two events:

- DRAM refresh requests
- DRAM requests from the processor

The DRAM controller (Figure 3) has four distinct blocks: control logic, address latches, address muxes, and the DRAM SIMM. This section describes each block.

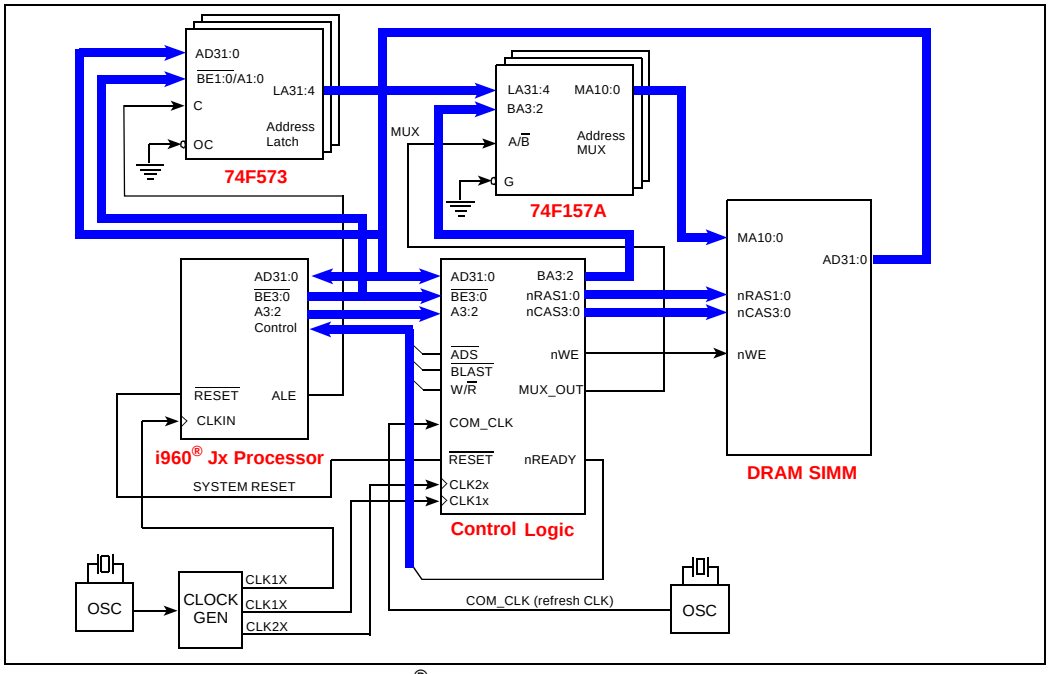

Figure 3. i960® Jx Processor DRAM Controller

2.1 Control Logic

The DRAM controller is centered around a four-bit state machine that controls accessing as well as refreshing of the DRAM banks. All signals are generated based on the four-bit state machine's outputs.

2.1.1 Refresh Logic

Typical 2 Mbyte x32 DRAMs need to be refreshed every 15.5 µs. The DRAM controller refreshes the banks using CAS-before-RAS refresh when the request is granted.

A refresh request has priority over a processor request. When a processor and a refresh request occur simultaneously, the DRAM controller sequences a refresh to the DRAM banks while the ACC_PENDING state machine posts the processor request. The pending request is then serviced after the refresh is done.

In the tested design, an eight-bit synchronous up-counter generates refresh requests. The counter is clocked using COM_CLK (7.3728 MHz clock). REFRESH_SYNC is asserted each time the counter reaches 98 (13.7 µs) which in turn asserts DRAM_REF. The REFRESH_SYNC equation synchronizes the output (when counter reaches 98) of the REFRESH state machine (which uses COM_CLK) to CLK1X. The refresh counter is reset to zero and counting resumes after the DRAM_STATES state machine completes servicing the refresh. During reset, the counter is also loaded with a zero.

2.1.2 Clock Generation

In this design a Motorola* MC88916D low skew CMOS PLL was used to generate the clock signals for the DRAM controller. The MC88916D uses the output of the oscillator as an input and outputs a 2X and 4X copy of the oscillator output:

- The 2X output is referred to as CLKIN or CLK1X
- The 4X output is referred to as CLK2X

The MC88916D produces a very low skew copy of both CLK1X and CLK2X. At 33 MHz, the maximum skew between the MC88916 and any of its outputs is ±1 ns, while the skew between any of the individual outputs is ±750 ps under equal loading conditions. In this design all clock lines are series terminated with 22-ohm resistors.

2.1.3 Wait State Profile

Table 1 provides typical wait state profiles for read and write accesses. the listed wait states have been tested with the design by changing the clock frequency and wait states and meeting the timing requirements for MUX, $\overline{RAS}$ and $\overline{CAS}$.

Table 1. i960® Jx Processor's Possible DRAM Wait State Combinations

DRAM Access Time	Mode	16 MHz Bus	20 MHz Bus	25 MHz Bus	33 MHz Bus	40 MHz Bus[1]
60ns	Read	1, 1, 1, 1	1, 1, 1, 1	1, 1, 1, 1	3, 1, 1, 1	3, 1, 1, 1
	Write	1, 0, 0, 0	1, 0, 0, 0	1, 1, 1, 1	2, 1, 1, 1	2, 1, 1, 1
70ns	Read	1, 1, 1, 1	1, 1, 1, 1	2, 1, 1, 1	3, 1, 1, 1	3, 2, 2, 2
	Write	1, 0, 0, 0	1, 0, 0, 0	1, 1, 1, 1	2, 1, 1, 1	2, 1, 1, 1
80ns	Read	1, 1, 1, 1	1, 1, 1, 1	2, 1, 1, 1	3, 1, 1, 1	4, 2, 2, 2
	Write	1, 0, 0, 0	1, 1, 1, 1	2, 1, 1, 1	3, 1, 1, 1	3, 2, 2, 2
100ns	Read	1, 1, 1, 1	2, 1, 1, 1	3, 1, 1, 1	4, 2, 2, 2	4, 3, 3, 3
	Write	1, 0, 0, 0	2, 1, 1, 1	3, 1, 1, 1	3, 1, 1, 1	4, 2, 2, 2

NOTE:
1. Contact your local Intel Sales Representative to determine availability of the 40 MHz i960 JA/JF processor.

2.2 Address MUX

Figure 4 is a block diagram of the latched DRAM address logic. The quad 2-input multiplexers transmit row and column addresses across the combined MA10:0 bus. The row address is passed through to the DRAM when $\overline{RAS}$ is asserted and the column address is passed through when $\overline{CAS}$ is asserted. The multiplexers are always enabled because the enable pins are tied low. BA3:2 signals are generated by using A3:2, respectively. BA3:2 are the only address bits that increment during a burst. The timing of these signals during bursts is critical for proper operation.

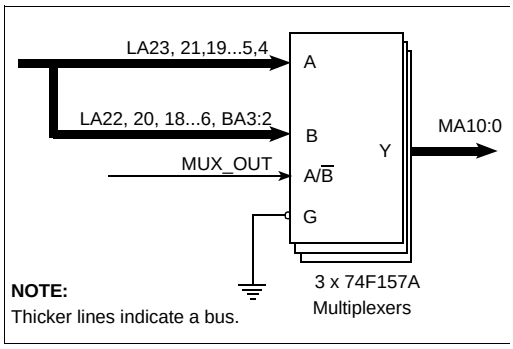

Figure 4. DRAM Address Multiplexers

2.3 Address Latch Path

As shown in Figure 5, the addresses are transferred to the latch outputs when the ALE signal is asserted. The latched addresses are always seen at the outputs by tying the output enables low.

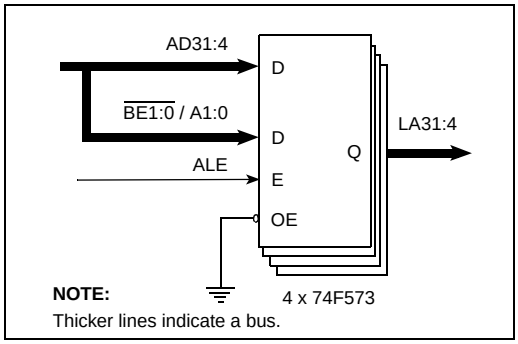

Figure 5. Address Latches

2.4 SIMMS

The SIMM block consists of one standard 72-pin SIMM socket, arranged as one or two banks. The parity bits of a x36 SIMM are not used in this design. However, x36 SIMMs are standard for PCs and workstations making them more readily available. The only penalty is more loading on the address and control lines due to the extra DRAM devices on the x36 SIMM.

In this design all address and control lines to the SIMM are series terminated with 22-ohm resistors.

3.0 STATE MACHINES AND SIGNALS

This section describes the state machines and signals used in this design. Most of the state machines are simple and the PLD equations can be referenced in APPENDIX A. The DRAM controller's state machine DRAM_STATES is the most complex of all the state machines; for that reason, this application note provides more detail on the operations of this state machine.

In this design, some of the state machines are clocked with the CLK1X clock (bus clock frequency), while others are clocked off the CLK2X clock (twice the bus clock frequency), or the COM_CLK (communications\UART clock frequency).

Certain state machines and their outputs were clocked off of CLK2X so that their outputs could use PH1 or PH2 as conditions for transition. The high portion of CLK1X is referenced as PH1 (phase one). The low portion of CLK1X is referenced as PH2 (phase 2).

All PLD equations are written in ABEL. APPENDIX A, PLD EQUATIONS, contains a listing of the PLD equations file. The state machine transitions described here follow the ABEL conventions for logic operators.

- ! represents NOT, bit-wise negation
- & represents AND
- # represents OR

In this design signals do not have a polarity assigned, signals that begin with a "n" signify that the signal is active low. For example, nRAS0 refers to the non-asserted state (SET_HIGH) and !nRAS0 refers to the asserted state (SET_LOW).

3.1 LATCH_ACC State Machine

The LATCH_ACC state machine is a one-bit state machine that monitors DRAM requests from the processor. This state machine is necessary because a DRAM request may occur during a refresh. DRAM refresh has priority over a processor's request. Therefore, this state machine is used to post the processor request. This state machine generates the LATCHACC signal.

3.2 DRAM_ACC Signal

The DRAM_ACC signal monitors DRAM requests from the processor, and pending DRAM requests from LATCH_ACC state machine. When conditions are met DRAM_ACC initiates the DRAM portion of the DRAM controller. The DRAM_ACC state machine generates the DRAMACC signal.

3.3 DRAM_BANK State Machine

The DRAM_BANK state machine is a one-bit state machine that decodes AD20 to select between DRAM bank 0 (!AD20) or DRAM bank 1 (AD20). This state machine is clocked using the CLK1X clock. The DRAM_BANK state machine generates the DRAMBANK signal.

DRAM bank decoding is dependent on which size DRAM module is used. The following address lines provide contiguous memory:

- AD20 is decoded for a 2 Mbyte SIMM
- AD22 is decoded for a 8 Mbyte SIMM
- AD24 is decoded for a 32 Mbyte SIMM
- For 1 Mbyte, 4 Mbyte and 16 Mbyte SIMM modules, only one bank is possible

The DRAM_BANK state machine is currently fixed for 2 Mbyte SIMM. If a size other than this default size is needed, replace AD20 with the appropriate address bit.

3.4 MUX State Machine

The MUX state machine is a one-bit state machine that is used to control the address multiplexers, essentially to select between row (MUX) or column addresses (!MUX). It is clocked using the CLK2X clock. This state machine generates the MUX_OUT signal.

3.5 BURST_ADDR State Machine

The BURST_ADDR state machine is a two-bit state machine that is toggled on burst accesses to select the next data word (next column data). The state machine is initially loaded with the value of the processor's A3:2 during the address state (IDLE) and then increments after each data state (DATA). This state machine is clocked using the CLK1X clock, and generates the BA3:2 signal. These signals are an input to the DRAM Address Muxes.

3.6 REFRESH State Machine

The REFRESH state machine is a seven-bit state machine that is used as a counter to indicate when a refresh is required. The state machine is clocked using COM_CLK (7.3728 MHz). In the tested system, COM_CLK was present to drive a UART for communications. This state machine sequences from 0 to 98 then resets to zero. Sequencing begins when DRAM_REF is not asserted.

3.7 REFRESH_SYNC Signal

The REFRESH_SYNC signal monitors the REFRESH state machine counter. REFRESH_SYNC is asserted when the counter reaches a count of 98. The REFRESH_SYNC equation synchronizes the output (when counter reaches 98) of the REFRESH state machine, which uses COM_CLK, to CLK1X.

3.8 DRAM_REF State Machine

The DRAM_REF state machine is a one-bit state machine that uses the output of the REFRESH_SYNC equation. This state machine is clocked using the CLK1X clock, and generates the DRAMREF signal. DRAMREF signals the DRAM_STATES machine that a refresh is needed. DRAMREF is deasserted when the refresh states complete.

3.9 nPRE_CAS State Machine

The nPRE_CAS state machine is a one-bit state machine and generates the nPRECAS signal. This signal is generated one CLK2X clock cycle earlier than nCAS3:0. The output of this state machine is then fed to the nCAS3:0 equations where it is asserted at the correct time. This state machine is clocked using the CLK2X clock.

3.10 nCAS3:0 Signals

The nCAS3:0 signals are asserted when nPRE_CAS and the respective nBE3:0 signal are asserted. The output of this state machine is then fed to the DRAM. This equation is clocked using the CLK2X clock.

3.11 nRAS0 State Machine

The nRAS0 state machine is a one-bit state machine that is used to generate the $\overline{RAS}$ signal for bank 0. This state machine is clocked using the CLK2X clock, and generates the nRAS_0 signal.

3.12 nRAS1 State Machine

The nRAS1 state machine is a one-bit state machine that is used to generate the $\overline{RAS}$ signal for bank 1. This state machine is clocked using the CLK2X clock, and generates the nRAS_1 signal.

3.13 nREADY Signal

The nREADY signal is asserted during the data cycle of a DRAM access. Data is valid on the rising edge of CLK1X while nREADY is asserted. $\overline{BLAST}$ from the processor deasserts on the rising edge of CLK1X if nREADY is asserted during the last data cycle of the request. nREADY is clocked using the CLK1X clock.

3.14 nWE State Machine

The nWE state machine is a two-bit state machine that is asserted while a DRAM write is in progress. It controls the $\overline{WE}$ lines of the banks to perform early writes. This state machine is clocked using the CLK2X clock, and generates the nWE_OUT signal.

4.0 DRAM CONTROLLER ACCESS FLOW

This section explains how the DRAM_STATES state machine is sequenced while reading, writing, and refreshing the DRAM. Examples used are:

- single-word read and write access
- quad-word read and write access
- refresh

Figure 6 shows the complete DRAM_STATES state machine for the tested 33 MHz design. More detail on the states and state machine are given in the following examples.

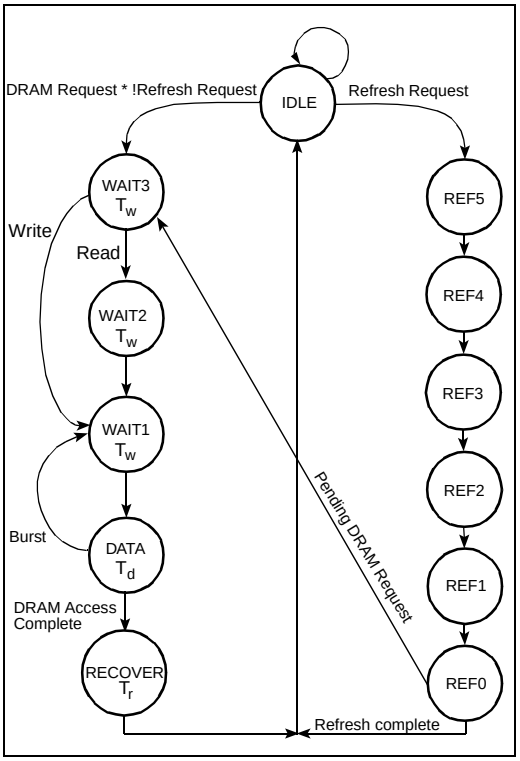

Figure 6. DRAM_STATES State Machine

4.1 Single-Word Access

Figure 7 shows the state diagram for a single-word read and write. Figure 8 is the single-word read timing diagram; Figure 9 is the single-word write timing diagram.

From the IDLE state the state machine waits for a DRAM access request or a refresh request. If a DRAM access is requested and not a refresh request, the state machine then transitions to the next state (WAIT3).

In WAIT3 state when PH1 is true, nRAS0 or nRAS1 is asserted depending on the bank selection; MUX is then asserted when PH2 is true. If the current access is a read, the machine proceeds to the WAIT2 state. If the current access is a write, the signals that normally assert in WAIT2 state (described below) now assert in WAIT3 state. The machine

proceeds to WAIT1 state bypassing WAIT2 state when the access is a write.

In WAIT2 state when PH2 is true, nCASx (x=3, 2, 1, or 0) is asserted if nPRE_CAS and the respective byte enable signal from the processor is asserted. The nWE signal is also asserted in WAIT2 state when PH2 is true. The machine then enters WAIT1 state.

When WAIT1 state is true, the respective nCAS signal is asserted when WAIT1 and PH2 are true. At this point, the processor has already asserted $\overline{\text{BLAST}}$. The machine then proceeds to the DATA state.

Data is valid in the DATA state (T_d) on the rising edge of CLK1X. When the DATA state is true, nREADY asserts. The respective nCAS signals deassert when DATA and PH2 are true. From the DATA state, the machine proceeds to the RECOVER state since $\overline{\text{BLAST}}$ is asserted.

When RECOVER is true, nREADY deasserts. When RECOVER and PH1 are true, MUX and the respective nRAS signals deassert completing the single word access.

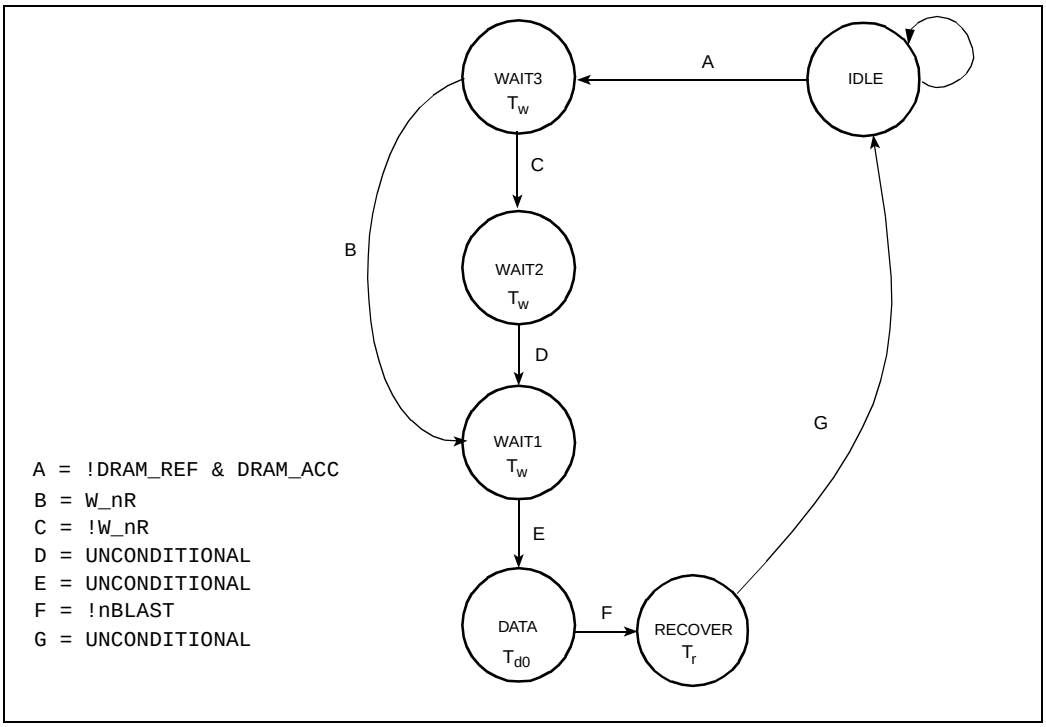

Figure 7. Single-Word Read and Write State Diagram (A2 = 1)

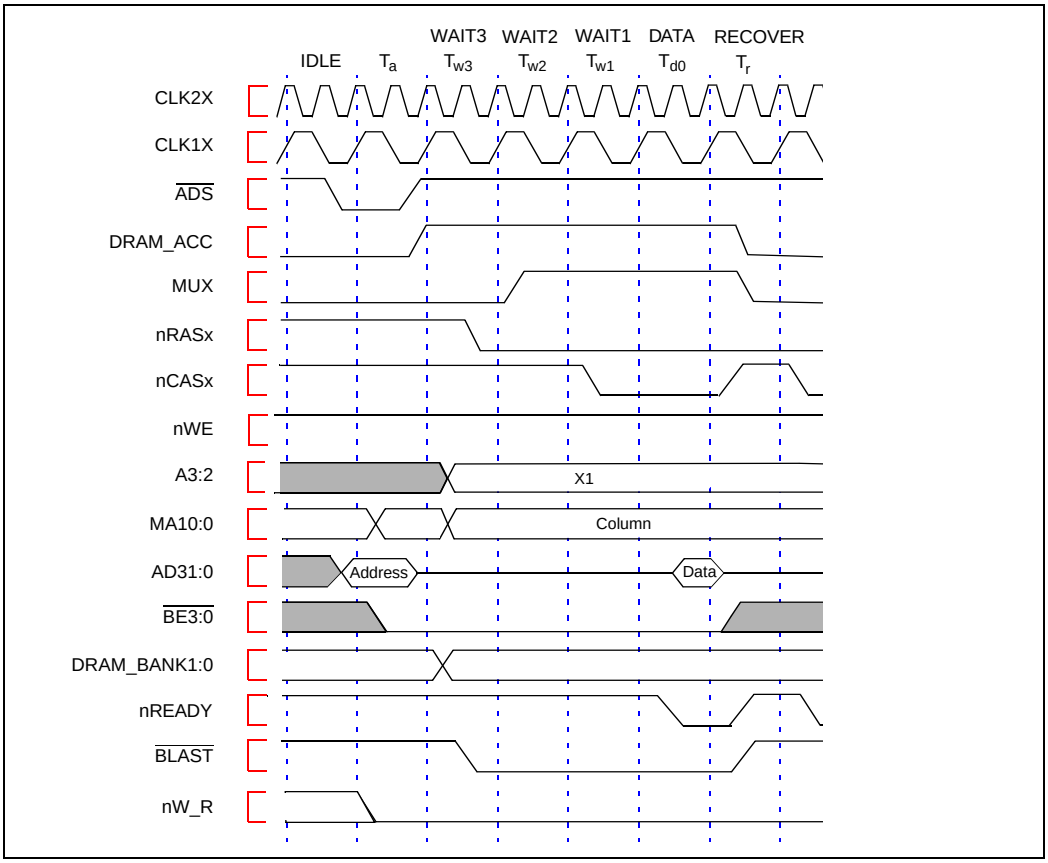

Figure 8. Single Word Read Timing Diagram

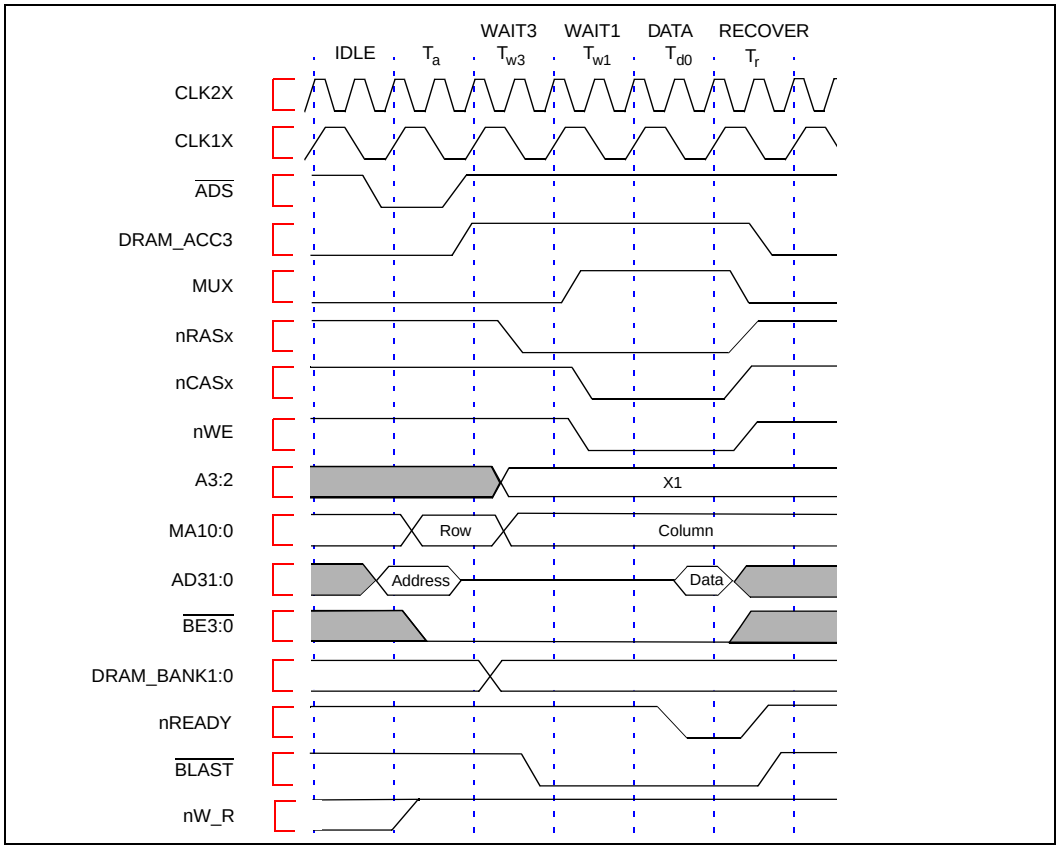

Figure 9. Single Word Write Timing Diagram

4.2 Quad-Word Access

Figure 10 shows the state diagram for a quad-word access. This state diagram also applies for triple-, and double-word accesses. Figure 11 is the quad-word read timing diagram; Figure 12 is the quad-word write timing diagram.

From the IDLE state the state machine waits for a DRAM access request or a refresh request. If a DRAM access is requested and not a refresh request, the state machine then transitions to the next state (WAIT3).

In WAIT3 state when PH1 is true, nRAS0 or nRAS1 is asserted depending on the bank selection; MUX is then asserted when PH2 is true. If the current access is a read, the machine proceeds to the WAIT2 state. If the current access is a write, the signals that normally assert in WAIT2 state (described below) assert in WAIT3 state. The machine proceeds to WAIT1 bypassing WAIT2 state when the access is a write (see Figure 10).

In WAIT2 state when PH2 is true, nCASx (x=3, 2, 1, or 0) is asserted if nPRE_CAS and the respective byte enable signal from the processor is asserted. The nWE signal is also asserted in WAIT2 state when PH2 is true. The machine then enters WAIT1 state.

From the WAIT1 state the machine proceeds to the DATA state.

The DATA state is the data cycle (T_d) of the access. Data is valid on the rising edge of CLK1X. When the DATA state is true, nREADY asserts. The respective nCAS signals deassert when DATA and PH2 are true. From the DATA

state, the machine can proceed to the RECOVER state if $\overline{BLAST}$ is asserted. If $\overline{BLAST}$ is not asserted, the state machine reenters WAIT1 state (see Figure 10).

When WAIT1 state is true, nREADY is deasserted. The respective nCAS signal is asserted when WAIT1 and PH2 are true. The machine then proceeds to the DATA state for the second data cycle (T_{d1}).

Data is valid in the DATA state (T_{d1}) on the rising edge of CLK1X. When the DATA state is true, nREADY asserts. The respective nCAS signals deassert when DATA and PH2 are true. From the DATA state, the machine can proceed to WAIT1 if $\overline{BLAST}$ is not asserted.

When WAIT1 state is true, nREADY is deasserted. The respective nCAS signal is asserted when WAIT1 and PH2 are true. The machine then proceeds to the DATA state for the third data cycle (T_{d2}).

Data is valid in the DATA state (T_{d2}) on the rising edge of CLK1X. When the DATA state is true, nREADY asserts. The respective nCAS signals deassert when DATA and PH2 are true. From the DATA state, the machine can proceed to WAIT1 if $\overline{BLAST}$ is not asserted.

When WAIT1 state is true, nREADY is deasserted. The respective nCAS signal is asserted when WAIT1 and PH2 are true. The processor asserts $\overline{BLAST}$ sometime in this state. The machine then proceeds to the DATA state for the fourth data cycle (T_{d3}).

Data is valid in the DATA state (T_{d3}) on the rising edge of CLK1X. When the DATA state is true, nREADY asserts. The respective nCAS signals deassert when DATA and PH2 are true. From the DATA state, the machine proceeds to the RECOVER state since $\overline{BLAST}$ is asserted.

When RECOVER is true, nREADY deasserts. When RECOVER and PH1 are true, MUX and the respective nRAS signals deassert, completing the quad word access.

The RECOVER state is a required state by the i960 Jx processor. This state is a function of the processor's $\overline{RDYRCV}$ pin. The processor continues to insert recovery states until it samples the $\overline{RDYRCV}$ pin HIGH. This function allows slow external devices more time to float their buffers before the processor drives address again. Transition to the next state, IDLE, is unconditional.

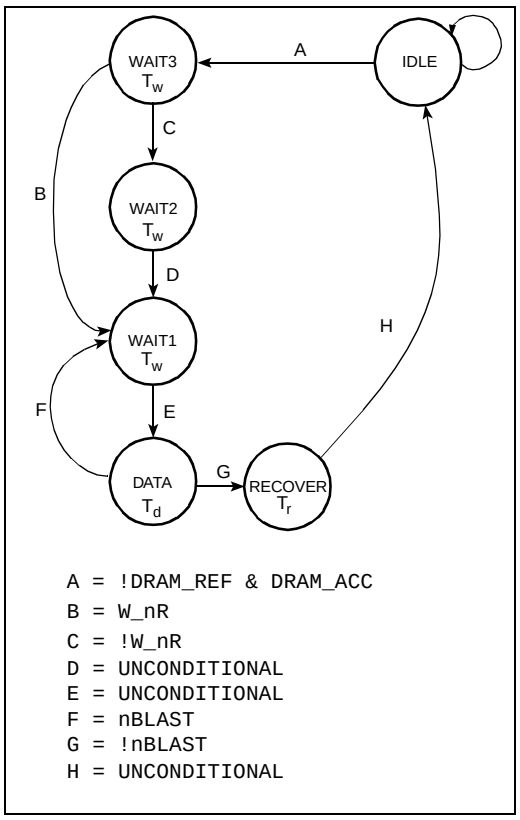

Figure 10. Quad-Word Read and Write State Diagram

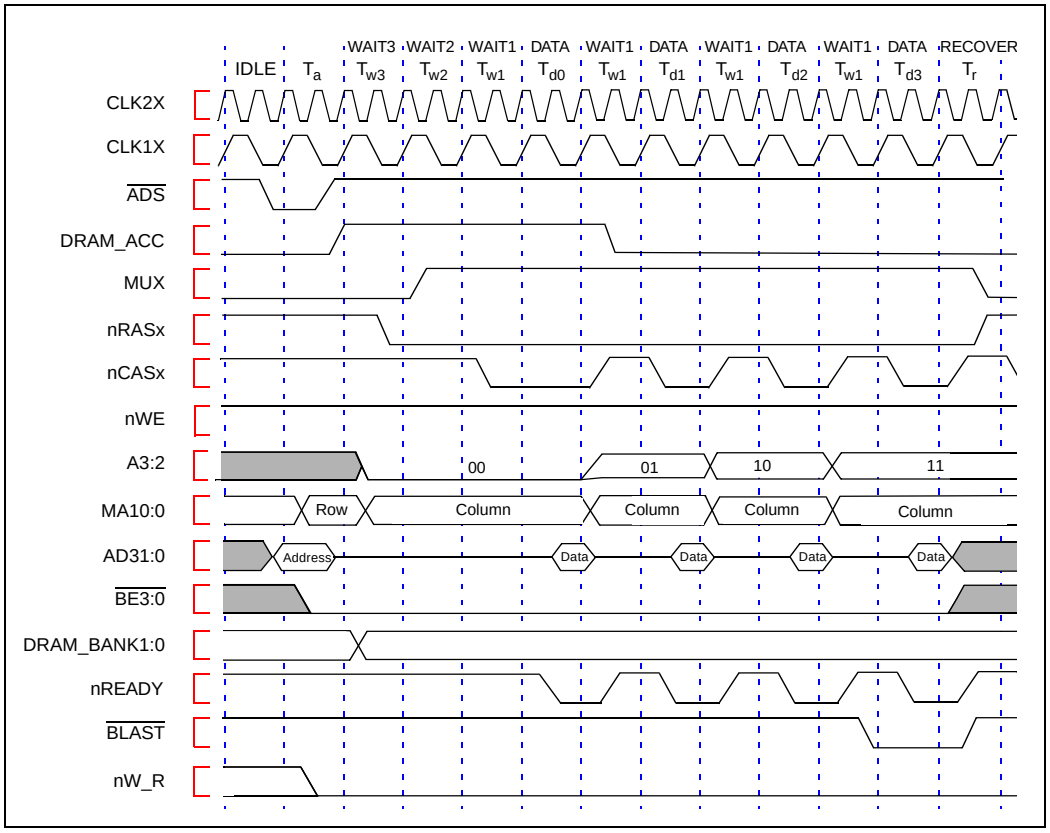

Figure 11. 3, 1, 1, 1 Quad-Word Read Timing Diagram

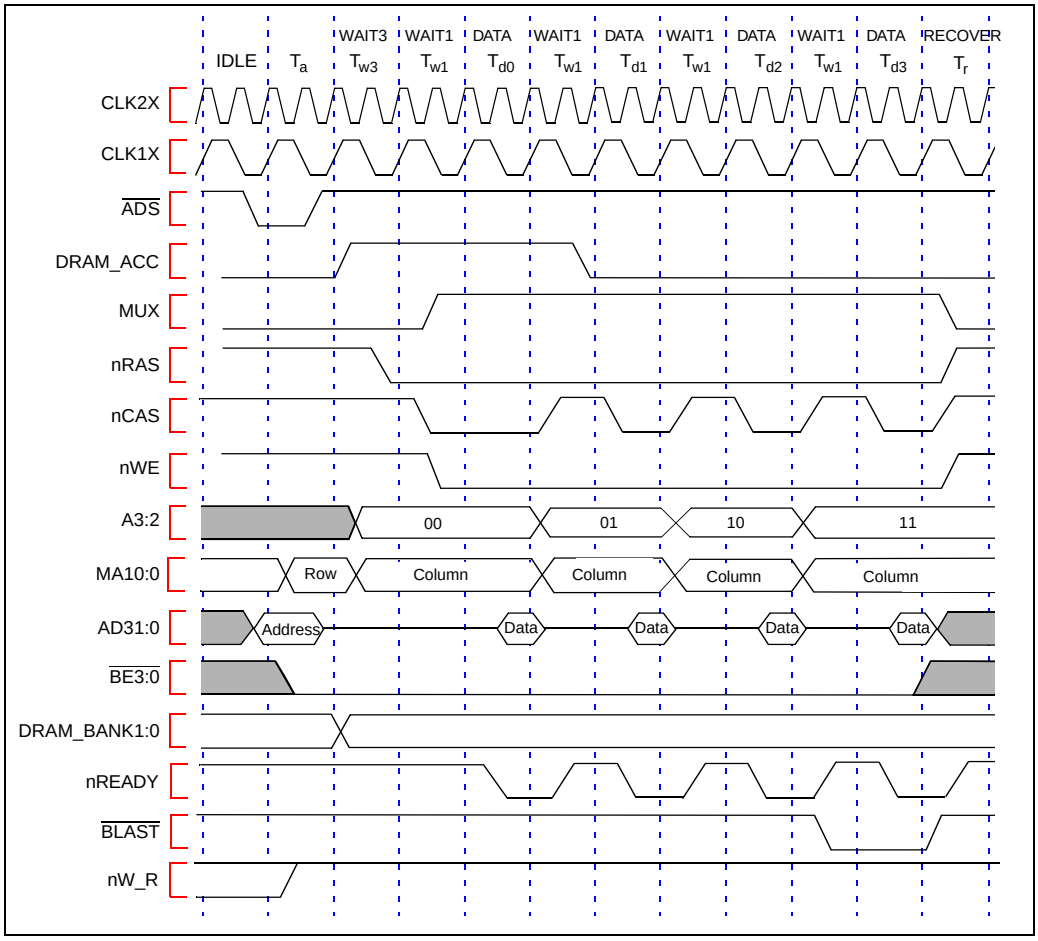

Figure 12. 2, 1, 1, 1 Quad-Word Write Timing Diagram

4.3 Refresh Cycles

In the tested design it was convenient to divide the communications\UART clock by 98 to provide a refresh request every 13.3 µs. The DRAM_STATES state machine services the request and sequences the refresh states. The example below describes the refresh states in the DRAM_STATES state machine.

Figure 13 shows the refresh state diagram; Figure 14 shows the refresh timing diagram with a pending DRAM request; Figure 15 shows the refresh timing diagram with no pending DRAM request.

In the IDLE state — if DRAMREF (see Section 3.8, DRAM_REF State Machine) is asserted and PH1 is true — all nCAS3:0 signals are asserted. Since DRAM_REF is asserted, the machine enters the REF5 state. The machine unconditionally proceeds to the REF4 state.

In the REF4 state when PH1 is true, the nRAS1:0 signals are asserted. The machine unconditionally proceeds to the REF3 state.

In the REF3 state when PH1 is true, the nCAS3:0 signals are deasserted. The machine unconditionally proceeds to the REF2 state.

In the REF2 state when PH2 is true, the nRAS1:0 signals are deasserted. The machine unconditionally proceeds to the REF1 state.

There are no signals that transition in state REF1. The machine unconditionally proceeds to the REF0 state.

When the REF0 state is true, the DRAMREF signal is deasserted. The machine then checks the DRAMACC status. If asserted, a DRAM access is pending and the machine proceeds to WAIT3 state (see Figure 13 and Figure 15). If DRAMACC is not asserted, the machine proceeds to the IDLE state (see Figure 13 and Figure 14).

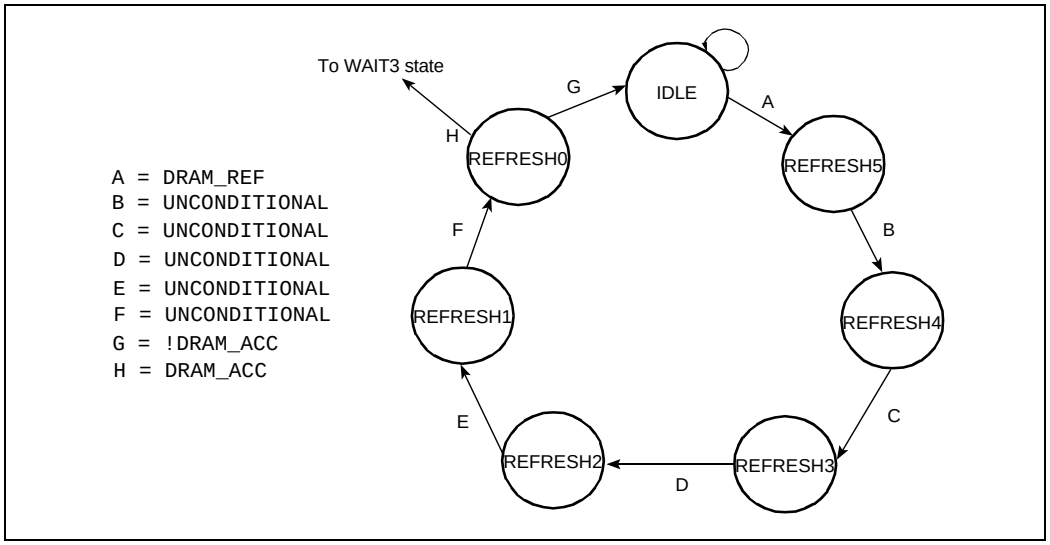

Figure 13. Refresh State Diagram

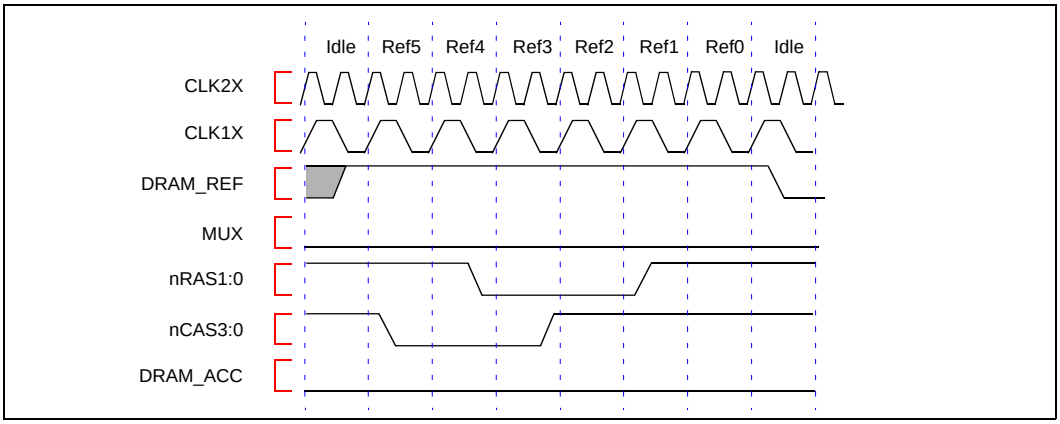

Figure 14. Refresh Timing Diagram (No Pending DRAM Request)

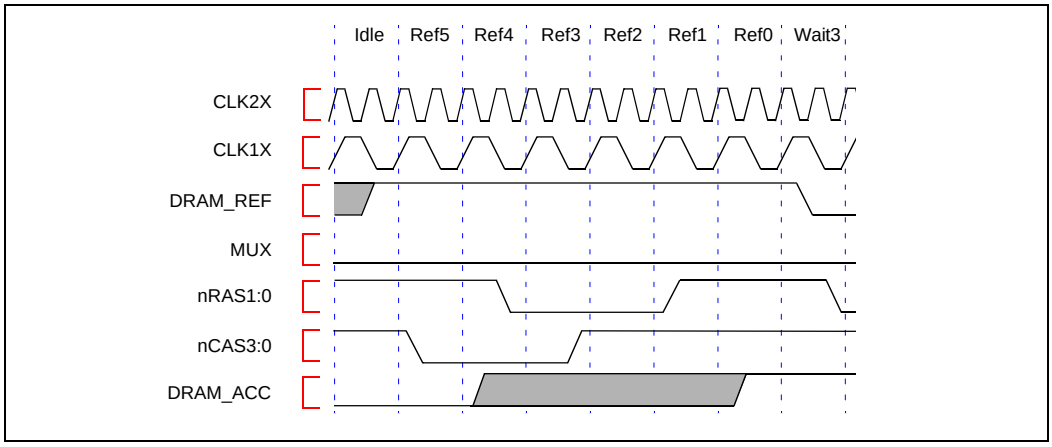

Figure 15. Refresh Timing Diagram (With Pending DRAM Request)

5.0 CONCLUSION

The DRAM Controller design successfully implements a single or dual bank 32-bit, non-interleaved DRAM controller. This design does this while using a minimum number of components. A standard 72-pin fast page mode DRAM SIMM was used to take advantage of the i960 Jx processor's burst bus capability. This design operates at 5 volts, with bus speeds up to 33 MHz. Various DRAM speeds, bus speeds up to 40 MHz, and wait state combinations were also successfully tested with minor changes to the basic design.

6.0 RELATED INFORMATION

This application note is one of four that are related to DRAM controllers for the i960 processors. The following table shows the documents and order numbers:

Document Name	App. Note #	Order #
DRAM Controller for the 33 MHz i960® CA/CF Microprocessors	AP-703	272627
Simple DRAM Controller for 25/16 MHz i960® CA/CF Microprocessors	AP-704	272628
DRAM Controller for the 40 MHz i960® CF Microprocessors	AP-706	272655

To receive these documents or any other available Intel literature, contact:

Intel Corporation
Literature Sales
P.O. Box 7641
Mt. Prospect IL 60056-7641
1-800-879-4683

To receive files that contain the timing analysis, schematics and PLD equations for this and the other DRAM controller application notes, contact:

Intel Corporation
America's Application Support BBS
916-356-3600

APPENDIX A
ABEL FILE

Table A-1 contains the PLD equations which were used to build and test the prototype design. Table A-2 defines signal and product term allocation. The PLD equations were created in ABEL* as a device-independent design. Using the ABEL software, a PDS file was created and then imported into PLDSHELL* software. PLDSHELL was used to "fit" the design into an Altera EPX780 FLEXlogic* PLD. PLDSHELL was also used to create the JEDEC file and to simulate the design.

Table A-1. i960® Jx Processor DRAM Controller ABEL File (Sheet 1 of 15)

```
Module          JXDRAM
Title           ' 80960 Jx DRAM Controller
Source File     JXDRAM.ABL
Revision        1.0

Date            11/15/94
Designer        Paul Durazo
                Intel 80960 Applications engineer'
" ******************************************************************************
" Non interleaved DRAM controller for the 80960JA/JF/JD with bus speeds up to 40MHz.
" One or two banks may be used depending on the DRAM size.
" See DRAM_BANK state machine for further information.
" DRAM start address is 0xA0000000
" DRAM wait state configuration is 3,1,1,1
" Active Low signals are designated with a (n) in front of the signal name.
" A (!) in front of a signal signifies it is cleared to 0.
" ie. nCAS is active low but not asserted, !nCAS is asserted or cleared to 0
" ******************************************************************************
"       Uxx     device  'iFX780_132';
" input signals
CLK1X           PIN; " 1X clock
CLK2X           PIN; " 2X clock
ICLK1X          PIN; " 1X clock fed to macrocell
COM_CLK         PIN; " Refresh  clock

AD31            PIN;
AD30            PIN;
AD29            PIN;
AD28            PIN;
AD22            PIN;
nADS            PIN;
W_nR            PIN;
nRESET          PIN;
nBLAST          PIN;
nBE3            PIN;
nBE2            PIN;
nBE1            PIN;
nBE0            PIN;
A3              PIN;
A2              PIN;
```

Table A-1. i960® Jx Processor DRAM Controller ABEL File (Sheet 2 of 15)

```
“output and node signals
nRAS_1          PIN     istype ‘reg’ ;
nRAS_0          PIN     istype ‘reg’ ;
nWE_OUT         PIN     istype ‘reg’ ;
MUX_OUT         PIN     istype ‘reg’ ;
BA3             PIN     istype ‘reg’ ;
BA2             PIN     istype ‘reg’ ;
DRAMREF         NODE    istype ‘reg’ ;
LATCH_ACC       NODE    istype ‘reg’ ;
DRAM_ACC        NODE    istype ‘com’ ;
DRAMBANK        NODE    istype ‘reg’ ;
REFRESH_SYNC    NODE    istype ‘reg’ ;
nREADY          PIN     istype ‘reg’ ;
nPRECAS         NODE    istype ‘reg’ ;
nCAS_3          PIN     istype ‘reg’ ;
nCAS_2          PIN     istype ‘reg’ ;
nCAS_1          PIN     istype ‘reg’ ;
nCAS_0          PIN     istype ‘reg’ ;
Q3              NODE    istype ‘reg’ ;
Q2              NODE    istype ‘reg’ ;
Q1              NODE    istype ‘reg’ ;
Q0              NODE    istype ‘reg’ ;
R6              NODE    istype ‘reg’ ; “Refresh counter bit 6
R5              NODE    istype ‘reg’ ; “Refresh counter bit 5
R4              NODE    istype ‘reg’ ; “Refresh counter bit 4
R3              NODE    istype ‘reg’ ; “Refresh counter bit 3
R2              NODE    istype ‘reg’ ; “Refresh counter bit 2
R1              NODE    istype ‘reg’ ; “Refresh counter bit 1
R0              NODE    istype ‘reg’ ; “Refresh counter bit 0

C               = .C.;
X               = .X.;

SET_LOW    = ^b0;
SET_HI     = ^b1;

“ Sate        |  I/O pin
“ references  |  references
PH1             = [ICLK1X];             “ high phase of 1X clock
PH2             = [!ICLK1X];            “ low phase of 1X clock
DRAM_STATES     = [Q3, Q2, Q1, Q0];     “ main dram state machine
nRAS0           = [nRAS_0];             “ Bank 0 RAS signal
nRAS1           = [nRAS_1];             “ Bank 1 RAS signal
nCAS3           = [nCAS_3];
nCAS2           = [nCAS_2];
nCAS1           = [nCAS_1];
nCAS0           = [nCAS_0];
MUX             = [MUX_OUT];
BURST_ADDR      = [BA3,BA2];
```

Table A-1. i960® Jx Processor DRAM Controller ABEL File (Sheet 3 of 15)

```
nWE             = [nWE_OUT];
DRAM_BANK       = [DRAMBANK];
DRAM_REF        = [DRAMREF];
LATCHACC        = [LATCH_ACC];
DRAMACC         = [DRAM_ACC];
REFRESH         = [R6, R5, R4, R3, R2, R1, R0];
nPRE_CAS        = [nPRECAS];

RECOVER = ^b0000;       " RECOVER is a recovery state and is required by the processor.
IDLE    = ^b0001;       " IDLE is the address state; determines if DRAM access has occurred.
WAIT3   = ^b0010;       " WAIT3-1 are wait states
WAIT2   = ^b0011;       "
WAIT1   = ^b0100;       "
DATA    = ^b0101;       " Data state
REF5    = ^b1000;       " REF5-1 are Refresh states
REF4    = ^b1001;       "
REF3    = ^b1010;       "
REF2    = ^b1011;       "
REF1    = ^b1100;       "
REF0    = ^b1101;       "

SA0 = ^b00;             " Burst address counter states
SA1 = ^b01;
SA2 = ^b10;
SA3 = ^b11;

RF0   = ^b0000000;      " Refresh counter States
RF1   = ^b0000001;
RF2   = ^b0000010;
RF3   = ^b0000011;
RF4   = ^b0000100;
RF5   = ^b0000101;
RF6   = ^b0000110;
RF7   = ^b0000111;
RF8   = ^b0001000;
RF9   = ^b0001001;
RF10  = ^b0001010;
RF11  = ^b0001011;
RF12  = ^b0001100;
RF13  = ^b0001101;
RF14  = ^b0001110;
RF15  = ^b0001111;
RF16  = ^b0010000;
RF17  = ^b0010001;
RF18  = ^b0010010;
RF19  = ^b0010011;
RF20  = ^b0010100;
RF21  = ^b0010101;
RF22  = ^b0010110;
RF23  = ^b0010111;
```

Table A-1. i960® Jx Processor DRAM Controller ABEL File (Sheet 4 of 15)

```
RF24  = ^b0011000;
RF25  = ^b0011001;
RF26  = ^b0011010;
RF27  = ^b0011011;
RF28  = ^b0011100;
RF29  = ^b0011101;
RF30  = ^b0011110;
RF31  = ^b0011111;
RF32  = ^b0100000;
RF33  = ^b0100001;
RF34  = ^b0100010;
RF35  = ^b0100011;
RF36  = ^b0100100;
RF37  = ^b0100101;
RF38  = ^b0100110;
RF39  = ^b0100111;
RF40  = ^b0101000;
RF41  = ^b0101001;
RF42  = ^b0101010;
RF43  = ^b0101011;
RF44  = ^b0101100;
RF45  = ^b0101101;
RF46  = ^b0101110;
RF47  = ^b0101111;
RF48  = ^b0110000;
RF49  = ^b0110001;
RF50  = ^b0110010;
RF51  = ^b0110011;
RF52  = ^b0110100;
RF53  = ^b0110101;
RF54  = ^b0110110;
RF55  = ^b0110111;
RF56  = ^b0111000;
RF57  = ^b0111001;
RF58  = ^b0111010;
RF59  = ^b0111011;
RF60  = ^b0111100;
RF61  = ^b0111101;
RF62  = ^b0111110;
RF63  = ^b0111111;
RF64  = ^b1000000;
RF65  = ^b1000001;
RF66  = ^b1000010;
RF67  = ^b1000011;
RF68  = ^b1000100;
RF69  = ^b1000101;
RF70  = ^b1000110;
RF71  = ^b1000111;
RF72  = ^b1001000;
```

Table A-1. i960® Jx Processor DRAM Controller ABEL File (Sheet 5 of 15)

```
RF73  = ^b1001001;
RF74  = ^b1001010;
RF75  = ^b1001011;
RF76  = ^b1001100;
RF77  = ^b1001101;
RF78  = ^b1001110;
RF79  = ^b1001111;
RF80  = ^b1010000;
RF81  = ^b1010001;
RF82  = ^b1010010;
RF83  = ^b1010011;
RF84  = ^b1010100;
RF85  = ^b1010101;
RF86  = ^b1010110;
RF87  = ^b1010111;
RF88  = ^b1011000;
RF89  = ^b1011001;
RF90  = ^b1011010;
RF91  = ^b1011011;
RF92  = ^b1011100;
RF93  = ^b1011101;
RF94  = ^b1011110;
RF95  = ^b1011111;
RF96  = ^b1100000;
RF97  = ^b1100001;
RF98  = ^b1100010;
RF99  = ^b1100011;
RF100 = ^b1100100;


"*****************************************************************************
"               DRAM STATE ASSIGNMENTS
"*****************************************************************************

state_diagram DRAM_STATES

state IDLE:                     " wait for a DRAM access or refresh request
        if(DRAM_REF) then REF5  " If refresh true begin refresh states
  else
      if(DRAMACC & !DRAM_REF) then WAIT3 " If DRAM access true continue to next state
  else
        IDLE;                   " Stay in IDLE state until refresh or DRAM access occurs

state WAIT3:                    " Tw, wait state 3
      if(W_nR) then WAIT1       " If a write skip the next state for 2,1,1,1 profile
  else
        WAIT2;                  " was a read so go to WAIT2 for 3,1,1,1 profile

state WAIT2:                    " Tw, wait state 2
        goto WAIT1;
```

Table A-1. i960® Jx Processor DRAM Controller ABEL File (Sheet 6 of 15)

```
state WAIT1:                    " Tw, wait state 1
        goto DATA;
state DATA:                     " Td, Data State
        if (nBLAST) then WAIT1  " Burst not complete go to WAIT1 for 1 wait state
  else
         RECOVER;               " Burst complete goto recovery state

state RECOVER:                  " Recovery State
        goto IDLE;

state REF5:
        goto REF4;              " Begin Dram Refresh, nCAS asserted coming into this state
state REF4:
        goto REF3;              " Refresh State, nRAS asserted in phase 1
state REF3:
        goto REF2;              " Refresh State, nCAS deasserted in phase 1
state REF2:
        goto REF1;              " Refresh State, nRAS deasserted in phase 2
state REF1:
        goto REF0;              "
state REF0:
        if(DRAMACC) then WAIT3  " Dram Access occured during refresh go to WAIT3 state
  else
        goto IDLE;              " No Dram Access pending go to IDLE state

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
"nRAS0 F/F, looks at Bank 0
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
state_diagram nRAS0

        state SET_LOW:

                if((DRAM_STATES == RECOVER) & PH1) then SET_HI
        else
                if((DRAM_STATES == REF2) & PH2) then SET_HI

        else
                SET_LOW;


        state SET_HI:

                if((DRAM_STATES == REF4) & PH1) then SET_LOW
        else
                if((DRAM_STATES == WAIT3) & PH1 & !DRAM_BANK) then SET_LOW

        else
                SET_HI;
```

Table A-1. i960® Jx Processor DRAM Controller ABEL File (Sheet 7 of 15)

```
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
"RAS1 F/F, looks at Bank 1
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
state_diagram nRAS1

        state SET_LOW:

                if((DRAM_STATES == RECOVER) & PH1) then SET_HI
        else
                if((DRAM_STATES == REF2) & PH2) then SET_HI

        else
                SET_LOW;

        state SET_HI:

                if((DRAM_STATES == REF4) & PH1) then SET_LOW
        else
                if((DRAM_STATES == WAIT3) & PH1 & DRAM_BANK) then SET_LOW

        else
                SET_HI;

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
" MUX F/F
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
state_diagram MUX
        state SET_HI:
                if((DRAM_STATES == RECOVER) & PH1) then SET_LOW  " deassert MUX
        else
                SET_HI;

        state SET_LOW:
                if((DRAM_STATES == WAIT3) & PH2) then SET_HI     " assert MUX
        else
                SET_LOW;
```

Table A-1. i960® Jx Processor DRAM Controller ABEL File (Sheet 8 of 15)

```
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
" Burst Address generated from A3:2,
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
state_diagram BURST_ADDR

STATE SA0:
        if(!nADS & !A3 & A2) then SA1     " load A3:2 address
else
        if(!nADS & A3 & !A2) then SA2     " load A3:2 address
else
        if(!nADS & A3 & A2) then SA3      " load A3:2 address
else
        if(nADS & !nREADY) then SA1       " increment A3:2 address
else
       SA0;                               " hold A3:2 address

STATE SA1:
        if(!nADS & !A3 & !A2) then SA0    "load A3:2 address
else
        if(!nADS & A3 & !A2) then SA2     "load A3:2 address
else
        if(!nADS & A3 & A2) then SA3      "load A3:2 address
else
        if(nADS & !nREADY) then SA2       "increment A3:2 address
else
       SA1;                               "hold A3:2 address

STATE SA2:
        if(!nADS & !A3 & !A2) then SA0    "load A3:2 address
else
        if(!nADS & !A3 & A2) then SA1     "load A3:2 address
else
        if(!nADS & A3 & A2) then SA3      "load A3:2 address
else
        if(nADS & !nREADY) then SA3       "increment A3:2 address
else
       SA2;                               "hold A3:2 address

STATE SA3:
        if(!nADS & !A3 & !A2) then SA0    "load A3:2 address
else
        if(!nADS & !A3 & A2) then SA1     "load A3:2 address
else
        if(!nADS & A3 & !A2) then SA2     "load A3:2 address
else
       SA3;                               "hold A3:2 address
```

Table A-1. i960® Jx Processor DRAM Controller ABEL File (Sheet 9 of 15)

```
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
"Write Enable F/F, asserts when MUX asserts; Deasserts when CAS goes invalid
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
state_diagram nWE

        state SET_LOW:
                if((DRAM_STATES == DATA) & !nBLAST) then SET_HI " deassert write enable
        else
                SET_LOW;

        state SET_HI:
                if((DRAM_STATES == WAIT3) & W_nR) then SET_LOW   " assert write enable
        else
                SET_HI;

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
" DRAM_BANK specifies which bank will be used (BANK 0 = !AD20) or (BANK 1 = AD20).
" For 2Mb simm decode AD20, 8Mb SIMM decode AD22, 32 Mb SIMM decode AD24.
" All other SIMM sizes only one bank is possible and nRAS1:
" 0 will have to be modified to reflect this.
" The default used in this design is a 2Mb SIMM which decodes AD20.
" If using a different SIMM size replace AD20 with the respective address bit
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
state_diagram DRAM_BANK
        state SET_HI:
                if(!nADS & AD31 & !AD30 & AD29 & !AD28 & !AD20)then SET_LOW "Bank0 sel
        else
                SET_HI;

        state SET_LOW:
                if(!nADS & AD31 & !AD30 & AD29 & !AD28 & AD20) then SET_HI "Bank1 sel
        else
                SET_LOW;

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
" DRAM Refresh Request F/F
" When refresh counter = 98, REFRESH_SYNC is set. DRAM_REF is cleared upon leaving
" the refresh states of DRAM_STATES state machine
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
state_diagram DRAM_REF
        state SET_HI:
                if(DRAM_STATES == REF0) then SET_LOW   " Refresh states are complete
        else
                SET_HI;

        state SET_LOW:
                if(REFRESH_SYNC) then SET_HI           " Refresh required
        else
                SET_LOW;
```

Table A-1. i960® Jx Processor DRAM Controller ABEL File (Sheet 10 of 15)

```
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
" Latch DRAM access
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
state_diagram LATCH_ACC
        state SET_HI:
                if(DRAM_STATES == DATA) then SET_LOW           "in DRAM state machine deassert
        else
                SET_HI;

        state SET_LOW:
                if(!nADS & AD31 & !AD30 & AD29 & !AD28) then SET_HI "DRAM access occurred
        else
                SET_LOW;

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
" DRAM Refresh counter
" The counter uses a 7.3728 MHz clock (COM_CLK). After 98 clocks (13.3 us) a refresh
" request (REFRESH_SYNC) bit is set.
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
state_diagram REFRESH
STATE RF0:
        if(!DRAMREF) then RF1
else
        RF0;
STATE RF1:
        goto RF2;
STATE RF2:
        goto RF3;
STATE RF3:
        goto RF4;
STATE RF4:
        goto RF5;
STATE RF5:
        goto RF6;
STATE RF6:
        goto RF7;
STATE RF7:
        goto RF8;
STATE RF8:
        goto RF9;
STATE RF9:
        goto RF10;
STATE RF10:
        goto RF11;
STATE RF11:
        goto RF12;
STATE RF12:
        goto RF13;
```

Table A-1. i960® Jx Processor DRAM Controller ABEL File (Sheet 11 of 15)

```
STATE RF13:
        goto RF14;
STATE RF14:
        goto RF15;
STATE RF15:
        goto RF16;
STATE RF16:
        goto RF17;
STATE RF17:
        goto RF18;
STATE RF18:
        goto RF19;
STATE RF19:
        goto RF20;
STATE RF20:
        goto RF21;
STATE RF21:
        goto RF22;
STATE RF22:
        goto RF23;
STATE RF23:
        goto RF24;
STATE RF24:
        goto RF25;
STATE RF25:
        goto RF26;
STATE RF26:
        goto RF27;
STATE RF27:
        goto RF28;
STATE RF28:
        goto RF29;
STATE RF29:
        goto RF30;
STATE RF30:
        goto RF31;
STATE RF31:
        goto RF32;
STATE RF32:
        goto RF33;
STATE RF33:
        goto RF34;
STATE RF34:
        goto RF35;
STATE RF35:
        goto RF36;
STATE RF36:
        goto RF37;
STATE RF37:
        goto RF38;
```

Table A-1. i960® Jx Processor DRAM Controller ABEL File (Sheet 12 of 15)

```
STATE RF38:
        goto RF39;
STATE RF39:
        goto RF40;
STATE RF40:
        goto RF41;
STATE RF41:
        goto RF42;
STATE RF42:
        goto RF43;
STATE RF43:
        goto RF44;
STATE RF44:
        goto RF45;
STATE RF45:
        goto RF46;
STATE RF46:
        goto RF47;
STATE RF47:
        goto RF48;
STATE RF48:
        goto RF49;
STATE RF49:
        goto RF50;
STATE RF50:
        goto RF51;
STATE RF51:
        goto RF52;
STATE RF52:
        goto RF53;
STATE RF53:
        goto RF54;
STATE RF54:
        goto RF55;
STATE RF55:
        goto RF56;
STATE RF56:
        goto RF57;
STATE RF57:
        goto RF58;
STATE RF58:
        goto RF59;
STATE RF59:
        goto RF60;
STATE RF60:
        goto RF61;
STATE RF61:
        goto RF62;
```

Table A-1. i960® Jx Processor DRAM Controller ABEL File (Sheet 13 of 15)

```
STATE RF62:
        goto RF63;
STATE RF63:
        goto RF64;
STATE RF64:
        goto RF65;
STATE RF65:
        goto RF66;
STATE RF66:
        goto RF67;
STATE RF67:
        goto RF68;
STATE RF68:
        goto RF69;
STATE RF69:
        goto RF70;
STATE RF70:
        goto RF71;
STATE RF71:
        goto RF72;
STATE RF72:
        goto RF73;
STATE RF73:
        goto RF74;
STATE RF74:
        goto RF75;
STATE RF75:
        goto RF76;
STATE RF76:
        goto RF77;
STATE RF77:
        goto RF78;
STATE RF78:
        goto RF79;
STATE RF79:
        goto RF80;
STATE RF80:
        goto RF81;
STATE RF81:
        goto RF82;
STATE RF82:
        goto RF83;
STATE RF83:
        goto RF84;
STATE RF84:
        goto RF85;
STATE RF85:
        goto RF86;
STATE RF86:
        goto RF87;
```

Table A-1. i960® Jx Processor DRAM Controller ABEL File (Sheet 14 of 15)

```
STATE RF87:
        goto RF88;
STATE RF88:
        goto RF89;
STATE RF89
        :goto RF90;
STATE RF90:
        goto RF91;
STATE RF91:
        goto RF92;
STATE RF92:
        goto RF93;
STATE RF93:
        goto RF94;
STATE RF94:
        goto RF95;
STATE RF95:
        goto RF96;
STATE RF96:
        goto RF97;
STATE RF97:
        goto RF98;
STATE RF98:
        goto RF0 ;
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
" PRE_CAS used for the nCAS3:0 signals to conserve on P-Terms.
" PRE_CAS is asserted one 2X clock before actual CAS signal asserts. This was done
" because PRE_CAS is routed through another macrocell to produce nCAS
" with its associated nBE.
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
state_diagram nPRE_CAS   "nREF_CAS

        state SET_LOW:

               if((DRAM_STATES == REF4) & PH2 & DRAM_REF) then SET_HI " REF. DEASSERT
        else
               if(DRAM_STATES == DATA) then SET_HI " CAS DEASSERT
        else
               SET_LOW;


        state SET_HI:

                if((DRAM_STATES == IDLE) & PH1 & DRAM_REF) then SET_LOW " REF. ASSERT
        else
                if((DRAM_STATES == WAIT2) & !W_nR) #(DRAM_STATES==WAIT1)then SET_LOW
        else

                SET_HI;
```

Table A-1. i960® Jx Processor DRAM Controller ABEL File (Sheet 15 of 15)

```
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
EQUATIONS

!nREADY := (DRAM_STATES == WAIT1); " DRAM ready
DRAMACC = LATCHACC # !nADS & AD31 & !AD30 & AD29 & ! AD28; " Flags DRAM accesses

!nCAS0 := !nPRE_CAS & (!nBE0 # DRAM_REF);

!nCAS1 := !nPRE_CAS & (!nBE1 # DRAM_REF);

!nCAS2 := !nPRE_CAS & (!nBE2 # DRAM_REF);

!nCAS3 := !nPRE_CAS & (!nBE3 # DRAM_REF);

REFRESH_SYNC := (REFRESH == RF98);    " Used to synch refresh counter to 1X clock

"*****************************************************************************
"               INITIALIZE CONDITIONS AND CLOCKING
"*****************************************************************************

[LATCH_ACC,REFRESH_SYNC,nREADY,DRAMREF,DRAMBANK,BA3,BA2,Q3,Q2,Q1,Q0,nWE_OUT].clk = CLK1X;
[LATCH_ACC, REFRESH_SYNC, DRAMREF, DRAMBANK, BA3, BA2, Q3, Q2, Q1, Q0].re = !nRESET;
[nREADY, nWE_OUT].pr = !nRESET;

[nPRE_CAS, nCAS3, nCAS2, nCAS1, nCAS0, nRAS_1, nRAS_0, MUX_OUT].clk = CLK2X;
[nPRE_CAS, nCAS3, nCAS2, nCAS1, nCAS0, nRAS_1, nRAS_0].pr = !nRESET;
MUX_OUT.re = !nRESET;

[R6, R5, R4, R3, R2, R1, R0].clk = COM_CLK;  " REFRESH COUNTER
[R6, R5, R4, R3, R2, R1, R0].re = !nRESET;

end JXDRAM
```

Table A-2. Signal and Product Term Allocation

OUTPUT MACROCELLS		BURIED MACROCELLS	
Signal	**Product Terms**	**Signal**	**Product Terms**
nRAS_1	4	nPRECAS	8
nRAS_0	4	DRAMREF	2
nWE_OUT	2	LATCH_ACC	5
MUX_OUT	2	DRAM_ACC	2
BA3	3	DRAMBANK	6
BA2	3	REFRESH_SYNC	1
nREADY	1	Q3	3
nCAS_3	2	Q2	4
nCAS_2	2	Q1	5
nCAS_1	2	Q0	5
nCAS_0	2	R6	3
		R5	7
		R4	6
		R3	5
		R2	4
		R1	5
		R0	6